1992

The Architecture of PDE Solving Systems

Elias N. Houstis
Purdue University, enh@cs.purdue.edu

John R. Rice
Purdue University, jrr@cs.purdue.edu

Report Number:
92-022

THE ARCHITECTURE OF PDE SOLVING SYSTEMS

Elias N. Houstis
John R. Rice

CSD-TR 92-022
April 1992

The Architecture of PDE Solving Systems*

Elias N. Houstis and John R. Rice
Computer Science Department
Purdue University
Technical Report CSD-TR-92-022
CAPO Report CER-92-10
April 20, 1992

Abstract

We present an architecture for high level systems for solving partial differential equations. The solution process is examined in some detail and components of future systems systematically identified. We discuss the principal non-numerical technical challenges: software integration and reuse, resource management, and expert system development. We also discuss some of the numerical/mathematical challenges found in creating future systems. Examples of editor components from Parallel ELLPACK are described.

1 INTRODUCTION

Within the next decade there will be enough computing power to support high level PDE solving systems for rather complex applications, i.e., involving a number of distinct domains and physical phenomena. An ideal system would have the following attributes:

- Geometry system for defining shapes and collections of domains,
- Interactive,
- Broad coverage, e.g., all second order PDEs, structural mechanics, multiple domains, simultaneous PDEs on domains, general 2D and 3D geometry,
- High level access to numerical and symbolic tools, methods of applied mathematics,
- Transparent access to high performance computing systems,
- Natural performance specifications, e.g., "obtain 3 digits of accuracy" and not "use a 39 by 68 mesh",

*This work supported in part by NSF grant CCR86-10817 and AFOSR grant F49620-92-I-0069.

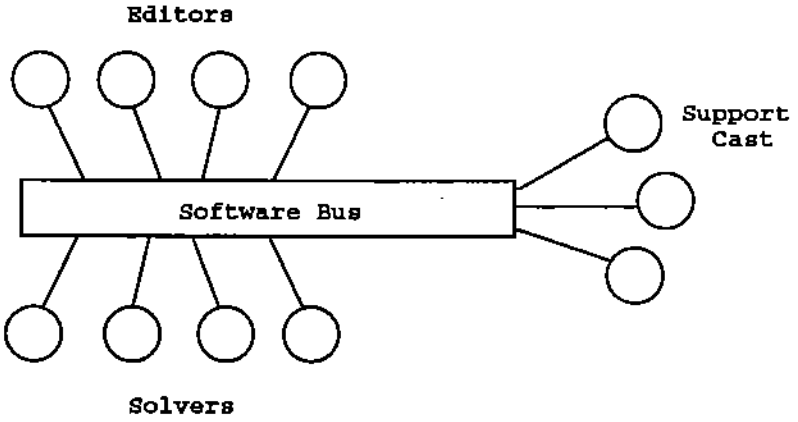


Figure 1: Schematic of the general architecture for a PSE.

Our thesis is that there is an evolutionary approach to building such systems which is well matched to the current rapid evolution of high performance computing. We systematically outline such an architecture and describe some of its components and technical challenges in detail. The ideas presented here are derived from earlier work on elliptic PDE solving systems with Ron Boisvert [9] and especially current collaborators in the Parallel ELLPACK project [5], [7]. An extended abstract of this paper appears in [6].

2 ARCHITECTURE

These systems are examples of problem solving environments (PSEs) and their general architecture is derived from that of PSEs. Space precludes a discussion of the various PSE architectures proposed in the literature; they have many elements in common and we assume the one whose schematic is given in Figure 1.

Editors are modules which interact with the user to receive or provide information. They might be quite simple (e.g., to specify the performance measures to be collected) or quite complex (e.g., a general 3D geometry modeler). *Solvers* are the engines which solve the underlying mathematical, geometric, or logical problems. They might be from very standard, low level libraries (e.g., Gauss elimination solver, expression parser) or huge systems (e.g., NASTRAN, Parallel ELLPACK). Note that a hierarchical structure will exist where editors have submodules which are solvers, or even complete PSEs, and solvers may have editors as submodules. The *support cast* consists of necessary modules which facilitate the computations. They may interact with editors or solvers or each other for tasks such as performance data collection, compilations, providing expert advice, etc.

The *software bus* is the key component, it provides the standard interfaces of the PSE and contains the data structures defining the global state of the computation. As editors, solvers, and support modules are added to a PSE one must construct interfaces between their internal structures and that of the software bus.

3 THE PDE SOLUTION PROCESS

We identify four general phases in the process of solving PDEs: problem *formulation*, *specification* of the solution method, *execution*, and *post-processing*. The four phases are further subdivided as shown in Figure 2 (formulation and specification of solution method) and Figure 3 (execution and postprocessing). These figures assume broad familiarity with solving PDEs and we clearly cannot elaborate on all the terms used. In interpreting these figures, keep in mind, for example, a system whose user is an automotive engineer and whose PDE problem encompasses every thing in an automobile engine. Thus, in Figure 2, the word *jargon* refers to the specialized terminology from an application area (e.g., piston, cam or coolant). There are PDEs for individual parts and there are many parts that are assembled together. Some parts have multiple PDEs which might be somewhat unrelated (e.g., those governing stress and temperature) and thus treated separately or the PDEs might be standard related sets (e.g., governing the three components of flow fields) treated as a unit. The terms *approximators* and *functionals* (linear ones) are used in their mathematical senses to express the myriad of ways to derive discretizations. This schematic recognizes the fact that almost all numerical methods for PDEs eventually involve a *linear equation solver* (even if only a diagonal system). The term *interface solvers* refers to techniques for solving systems of PDEs defined in separate but neighboring domains. The analysis of such solvers is still quite immature, the best developed subset is that of Schur complement techniques used in domain decomposition methods. In Section 5 we describe an editor for the equation specification subtask.

Figure 3 shows a collection of tasks that are primarily generic in nature. The principal exception is the validation process which is to provide confidence that the computed solution of the PDEs actually satisfies the accuracy requirements. This is one of the less developed aspects of solving PDEs and is discussed further in Section 7.

Note that the process illustrated by Figures 2 and 3 is quite static. In some applications the structure of the overall problem changes dynamically and thus the mix of components might not be as fixed or sequentially ordered as these figures suggest, see Section 7.

Figure 4 lists the principal components of the support cast for a PDE solving system. The four top level groups are generic to all PSEs but most of the component modules are specific to PDE systems.

4 THE PRINCIPAL TECHNICAL CHALLENGES

The PDE solving system envisaged here is larger and more complex than any existing scientific/engineering application system. Yet its construction does not assume new breakthroughs in technology, just substantial advances in a variety of areas. We divide the principal technical challenges into two groups, those generic for PSE systems and those specific to PDE solving systems.

4.1 Technical Challenges for PSEs

We identify four challenges as follows:

FORMULATION.	
SINGLE PDE.	
Geometry:	Modeler + Catalogs + Files
Equation:	Mathematics + Jargon
Conditions:	Mathematics + Jargon
COMPOSITION.	
Multiple PDEs:	Domain cloning + Multiple PDE solvers
Multiple Domains:	Assembly of objects
SOLUTION METHOD.	
SINGLE PDE.	
DISCRETIZATION.	
Approximators:	Basis functions + Meshes
Functionals:	Interpolation, least squares, energy, ...
TRANSFORMATION.	Symbolic manipulation, reordering of equations, changes of coordinates or variables, ...
ITERATION.	For nonlinear problems or faster solutions, Newton's method, SOR, conjugant gradient, ...
LINEAR SOLVER.	Gauss elimination, tridiagonal solvers (ADI), diagonal solvers (explicit time steps), triangular solvers, ...
MULTIPLE PDES.	
MULTIPLE SOLVERS:	Same as single PDE
INTERFACE SOLVERS:	Interaction at PDE/domain level, Merging of discrete problems, Direct solvers across interfaces

Figure 2: The subdivisions of the tasks of problem formulation and solution method specification for solving PDEs.

EXECUTION.
- MACHINE SELECTION.
  - Problem
  - Partitioning
- ERROR MONITORING.
- PERFORMANCE MONITORING.

POST-PROCESSING.
- VALIDATION OF SOLUTION
  - Comparisons: Previous solutions + New computations
  - Error Estimates: Residuals + Special formulas
- VISUALIZATION
- PERFORMANCE DISPLAYS
- SAVE FILES
- EXPORT RESULTS

Figure 3: The subdivision of the tasks of execution and post-processing for solving PDEs.

RESOURCE MANAGEMENT.
- Selection of Machines
- Problem Partitioning

PERFORMANCE MONITORING.
- Detection of Errors
- Detection of Unsatisfactory Performance
- Database of Performance Measures

EXTERNAL SYSTEM ACCESS.
- File and Storage Systems
- Escapes to Unrelated Systems

EXPERT ASSISTANCE.
- Problem Consistency: Geometry + Mathematics
- Method Selection: At all subphases + Overall
- Machine Selection: Problem sizing + Availability analysis
- Problem Partitioning:
  - For Parallel Computation
  - For Different Behaviors
- Validation

Figure 4: The principal components of the support cast for a PSE to solve PDEs.

Effective PSE Architecture. Figure 1 illustrates one plausible architecture but none of those proposed has been tested in even a prototype system, never mind a realistic one. The demands on the architecture are great, it must provide a high level of abstraction as the application coverage, methods and machines evolve over time in unanticipated ways. Thus the structure must remain stable while its foundations are changing. For PDE solving systems we believe the architecture will be geometry centered as this appears to be the most stable and central aspect of the application. The physical and mathematical problems will be attached to the geometrical objects and, in turn, solution methods attached to these problems.

Software Integration and Reuse. The cost of developing all the PDE solving software and encompassing system from scratch would be enormous and is prohibitive. The only practical approach is to integrate existing software libraries and specialized systems. Thus a major challenge for the architecture is the engineering of a high level software structure into which the existing software can be incorporated with reasonable cost. Essentially, one must transform the "world view" of the existing software to and from that of the PSE.

Computing Resources and Parallelism. The computing power required for this PSE is enormous but well within the reach of parallel supercomputers to become available soon. This means that the software and problem solving complexities are compounded by the necessity of handling parallelism. Further, the complexities of parallel computation should be hidden as much as possible from the user. This is probably infeasible to do this completely with the current state of the art, but it is long term goal of the architecture. We feel that the most effective problem partitioning method for PDE and many other PSEs will be geometry based, that is, one subdivides the physical domain so that each piece has approximately the same computing requirements.

Expert System Assistance. PSEs are very complex and yet this complexity must be mostly hidden from the user. This is to be accomplished using expert systems. Most expert systems are problem specific but so many are required that a systematic approach is needed for their creation and use. Note that PSEs have the ability to collect data on the problems they solve and this allows one to create and enlarge databases of knowledge (features, performances, results). The expert system methodology used should exploit this potential.

4.2 Technical Challenges Specific to PDEs

We do not discuss the well known and very difficult challenges of devising better algorithms or exploiting parallelism in numerical methods. Among the other challenges specific to PDE solving we note the following:

Validation and Accuracy Control. A key requirement for a PSE is the need to provide confidence that an adequate solution has been computed. The common methodology for validation is both expensive and of doubtful reliability: one solves the PDE a second time with a different method, a finer mesh, more terms, etc. A more sophisticated methodology is used in adaptive grid methods [1], [2] and this can be (but rarely is) used for solution validation. Closely related to validation is the problem of selecting mesh or grid sizes to meet accuracy specifications. Adaptive mesh methods can do this well for some PDEs (e.g., parabolic) but not all (e.g., elliptic). Both expert system and "computing rough trial solutions" are strategies that appear promising but which have not been proven in practice.

Problem Features. There are features of PDE problems known to be related to the choice of methods and difficulty of solution, e.g., boundary layers, singular perturbations, oscillatory, re-entrant corners. A large list of these is included in [3] (see also Appendix A of [9]). Many of these features are poorly defined or subjectively measured. Some of them can be determined easily if the functions involved have simple symbolic definitions. Measures for these features should be quantified for use in expert systems to aid in the solution process and we need robust, efficient algorithms to measure them independently of the particular form in which the functions are given (e.g., as a 10 character expression or as a 10,000 instruction object code).

High Level PDE Solvers. Almost all PDE solving methods are derived, viewed, and analyzed as techniques which solve a system of (algebraic) equations for a large set of values. These values represent in some way an approximate solution to the PDE problem. More study is needed of methods whose atomic step is solving some PDE for its solution (viewed purely as a function). A simple example of a high level approach is to solve a nonlinear PDE by linearizing it (by Frechet derivatives) to obtain a linear PDE which is solved at each step of Newton's method. No discretization is visible in this formulation of the high level method and the convergence, etc., is independent of how the linearized PDE is solved provided it is done accurately. We have reached the stage where most well behaved PDEs on simple domains can be solved efficiently and reliably. We should use this solution process as a building block for solving more complex problems instead of always working down to equations which involve large sets of coefficient or function values.

5 EDITORS

Space precludes describing many of the editors seen in Figures 2, 3, and 4, so we select two examples from Parallel ELLPACK to illustrate the nature of such editors. These are for equation specification and domain partitioning and, though well advanced, even they are not yet at the level of sophistication envisaged for the future.

5.1 Equation Specification

The equation specification editor window of Parallel ELLPACK is shown in Figure 5. The user interface is implemented using X-windows, its primary function is to allow one to specify the PDE operator and right side of the equation. The PDE solution is automatically named U and the derivative terms U_{xx}, etc. are given to prompt the user for a linear problem. A nonlinear PDE problem can also be specified (as seen in Figure 5). This editor provides symbolic facilities (MACSYMA in this case) to manipulate the mathematical form of the equation. Special symbolic facilities built into the editor are:

1. *PDE Linearization.* The Frechet derivative of the PDE operator is computed so that an iteration (Newton's method here) can be used to solve the PDE using the linearized equation.

2. *Solution Specification.* When testing and comparing PDE methods it is often very helpful to specify the solution $U(x, y)$ to be a known function with properties of interest. This is achieved in this editor by writing the solution in the "known solution" box.

5.2 Domain Partitioning

The domain partitioning editor helps a user subdivide the PDE domain into k pieces suitable for using k processors in parallel. At some future time this editor should become a solver, but parallel computation is not so well understood now and user interaction is usually needed. The objectives of the partitioning are, in geometric terms, to have pieces of equal sizes (containing equal numbers of grid points or elements), to have small perimeters, and to have few neighbors. To optimize these quantities exactly is infeasible so various heuristic methods are used. This editor is shown in Figure 6 and the buttons at the lower left select different heuristics. The partition can be edited by the user (points changed from one color to another) and quantitative measures of the objectives are displayed.

This editor uses a discretization of the domain for its partition, but it need not be the same discretization as used to approximate the PDE operator, it can be much coarser. The computation to partition, even approximately, a general domain of 10,000 elements into, say, 128 subdomains can be very long. It can take several hours on a rather fast workstation even using the better heuristic methods.

These editors record their results in the software bus of Parallel ELLPACK which is a script being constructed for eventual execution. The amount of information can be very large.

6 SOLVERS AND SUPPORT CAST

We only discuss one sample topic here, the selection of solution methods. Parallel ELLPACK currently just presents a menu (see Figure 5, at the right) for selecting numerical methods. An expert system approach is under development called PYTHIA [8] (formally called Athena [4]) and Figure 7 shows its "consultation window". The problem characteristics are determined from the PDE specification or provided by the user. The user provides the solution objectives. For example, one might want to run on the Intel iPSC/860 with an 81 by 81 grid, or one might want to achieve an accuracy of 0.6% and have the computation finished by next Monday morning.

The current elementary method selection facilities of Parallel ELLPACK are symptoms of the fact that this area is still very poorly understood. For example, most numerical analysts equate efficiency in solving PDEs to efficiency in solving linear systems of equations even though there is ample evidence that other factors are much more important than fast linear equation solvers. For example, very little thought and less analysis has been given to high level PDE solvers (see Section 4.2). For example, discussions of methods and problems invariably include descriptions of problem features (see Section 4.2), but we are aware of no study to define or measure such features scientifically. The study of the architecture of PDE solving systems has the beneficial effect of explicitly exposing this gap in our knowledge. Unfortunately, it will take some time to fill it.

We envisage a solver based on a taxonomy of PDE solving methods (not linear equation solving methods). It is supported by a support cast module (an expert system) which collects problem information and uses that in conjunction with its knowledge base to select appropriate methods. The support cast module will have the possibility of user interaction. For example, it must allow one to force the use of particular method even if it appears inferior. Similarly, a sophisticated user

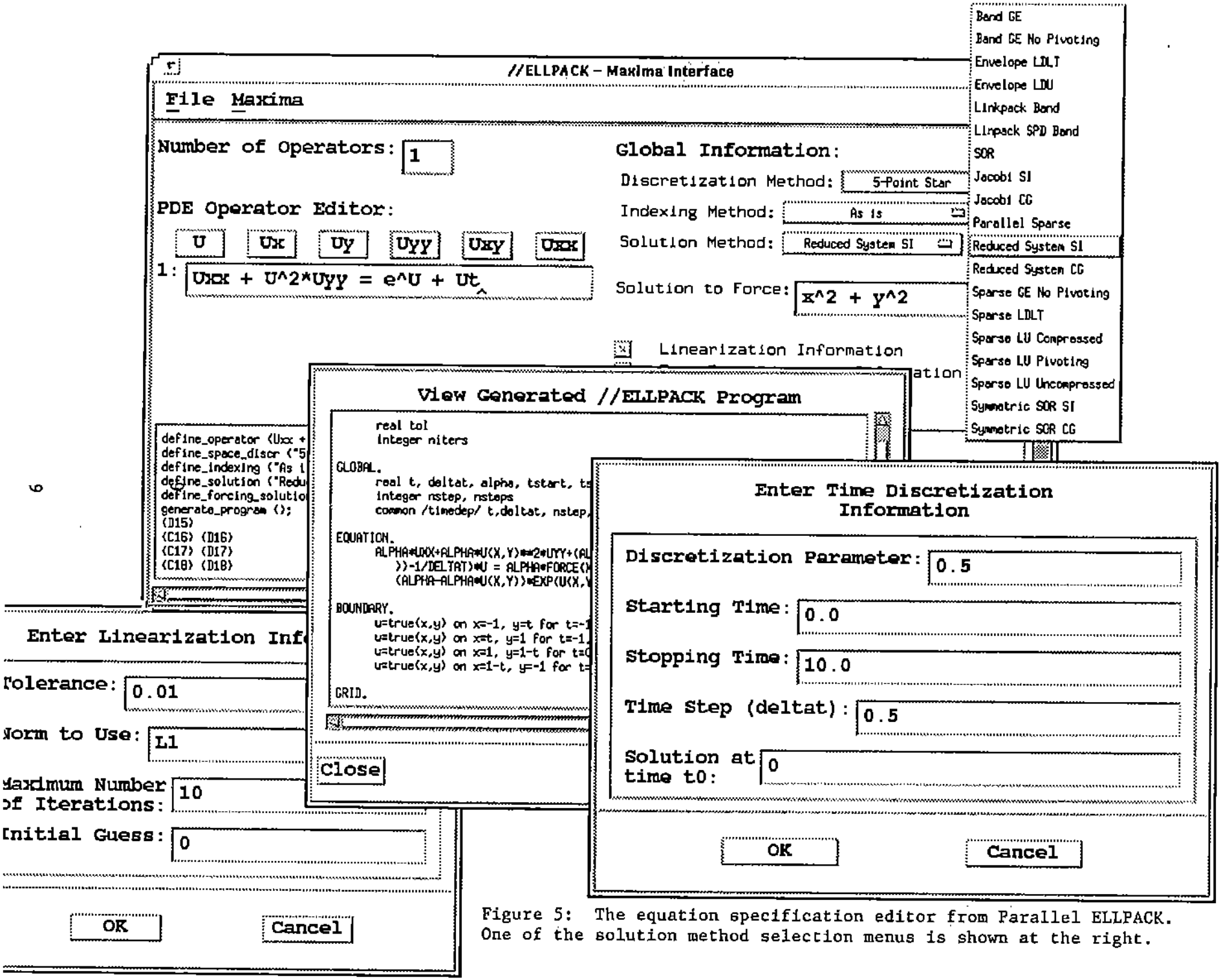


Figure 5: The equation specification editor from Parallel ELLPACK. One of the solution method selection menus is shown at the right.

Figure 6: The domain partitioning editor from Parallel ELLPACK.

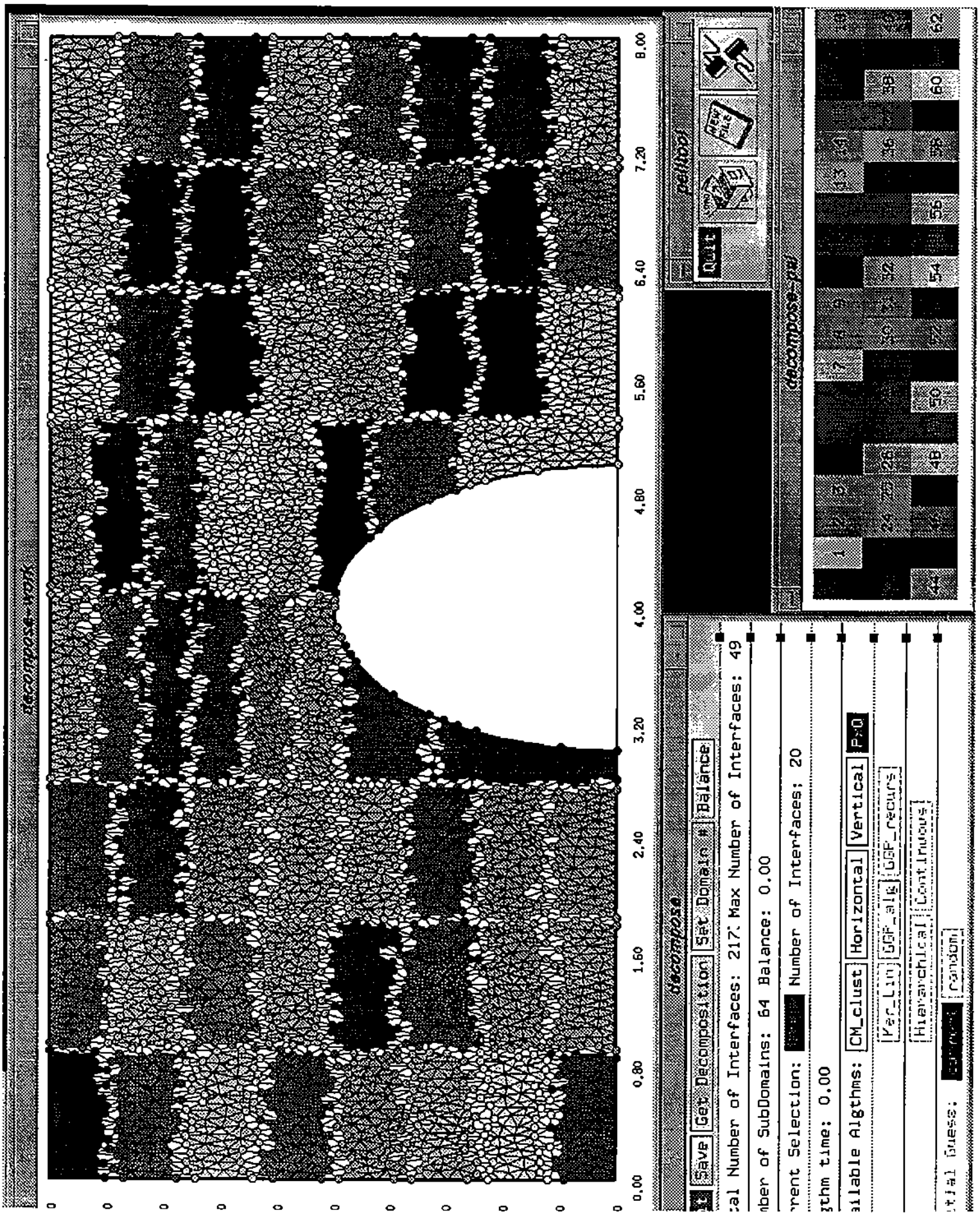
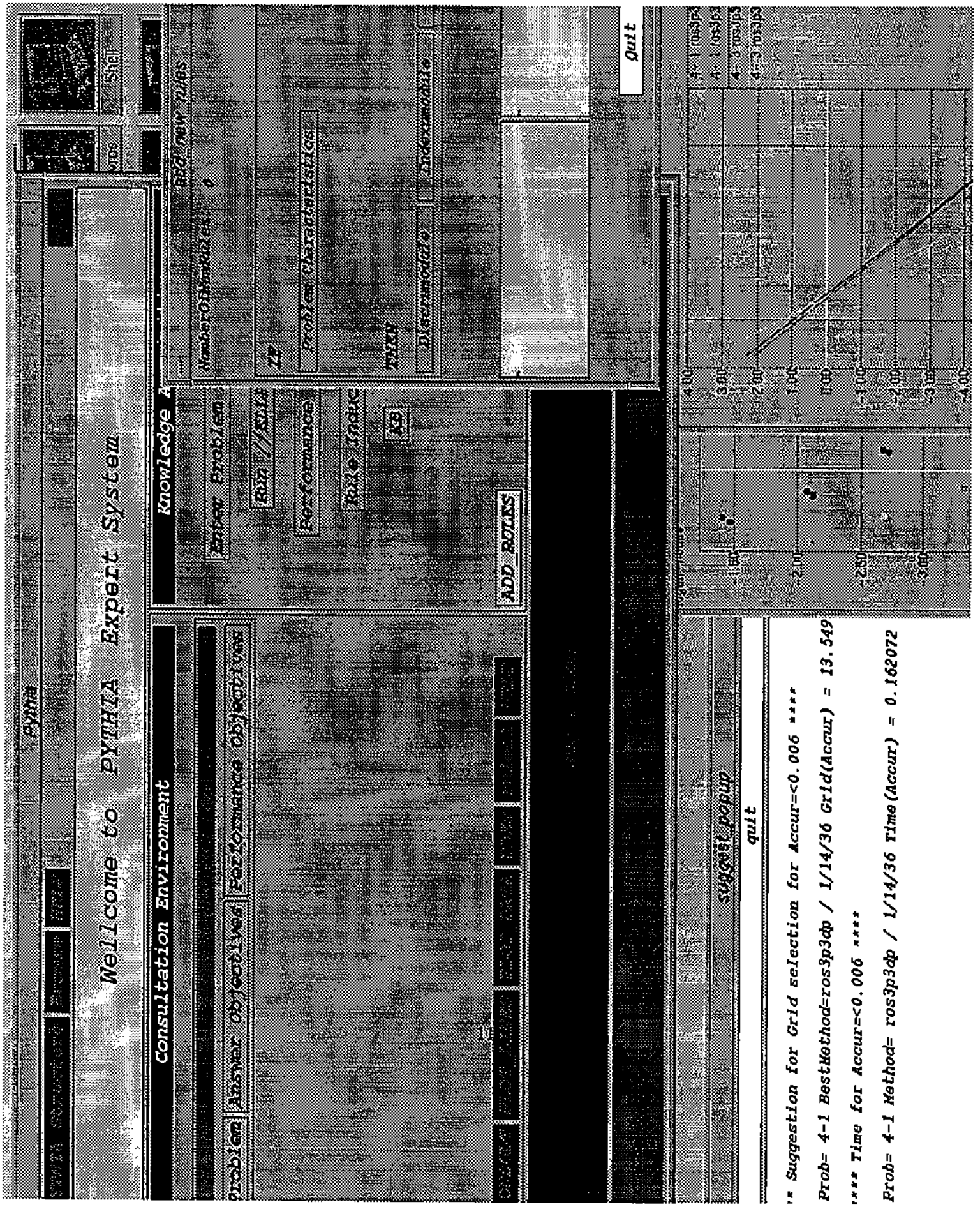


Figure 7: The consultation window from the PYTHIA system which will become the method selection editor for Parallel ELLPACK.



might know that the PDE at hand has a singularity which will not be discovered by the system, or perhaps discovered only at great expense.

7 OTHER CONSIDERATIONS

The modular structure of a PDE solving system was justified earlier primarily by software engineering issues (e.g., cost and evolution) and a static modular structure presented in Section 3. We elaborate on the earlier observations that the PDE solution process can be very dynamic. There is almost no characteristic of the process that does not change dynamically in some important application. The number of geometric domains can increase or decrease as physical parts break apart, as new phenomena appear (turbulence, shock waves, external sources). The nature of the PDE can change: from elliptic to hyperbolic in flows or new terms are added as new phenomena become important. Massive new subcomputations can appear: imagine simulation of the temperature in a ship deck from solar radiation when a bomb hits. The time scale may go from minutes to microseconds, the space discretization meshes may go from 10 inches to 10 thousandths of an inch. Some of these dynamic effects may be directed by the user, others may result in queries to the user ("Did you really mean to drop a bomb on my deck?", "and increase the computing bill from $2.13 to over $27,000?"). Many of these changes must be handled by a reconfiguration of the system state and processes.

The modular structure is also required by the large size of the computations. Many applications involve several users over long periods of time, a new device design is not completed in one run or in one afternoon session. One must allow for something like the old batch term *separate compilation.* The interactive work of one person for an hour or a day must be collected together for later use without repeating all the interactions. During a day, an engineer may find several "states" of the PDE solving system of interest and want to "save" them for later consideration. This facility must be available. Note that these activities may require very large storage facilities. A simple minded copying of everything relevant could result in many huge files being saved. This effect will no doubt motivate work in how to save the information more concisely. The other extreme is to save all the user input (keystrokes, voice commands, pointings). These files cannot be very large because people cannot generate quickly megabytes of information themselves. The choice here becomes a classic tradeoff between time and space.

Finally, we note that the requirement to save things implies that all the editors are "invertible". That is, whatever information is saved when the "save" button is pushed must be adequate to recreate the editor state if that information is presented to the editor again.

References

[1] D.C. Arney and J.E. Flaherty, An adaptive mesh-moving and local refinement method for time dependent partial differential equations, *ACM Trans. Math. Software*, **16**, (1990), 48–71.

[2] I. Babuska, O.C. Zienkiewiez, and J.E. Flaherty, *Accuracy Estimates and Adaptive Refinements in Finite Element Computations*, John Wiley, Chinchester, (1986).

[3] W.R. Dyksen, E.N. Houstis, and J.R. Rice, A population of second order, linear, elliptic problems, *Math. Comp.*, **36**, (1982), 475–484.

[4] C.E. Houstis, E.N. Houstis, J.R. Rice, P. Vavodoglou, and T.S. Papatheodorou, ATHENA: A knowledge base system for Parallel ELLPACK, in *Symbolic-Numeric Data Analysis and Learning*, (E. Diday and Y. Lechevallier, eds.), Nova Science Pub., New York, (1992), 459–467.

[5] Elias N. Houstis, T.S. Papatheodorou, and John R. Rice, Parallel ELLPACK: An expert system for parallel processing of partial differential equations, *Math. Comp. Simulation*, **31**, (1989), 497–508.

[6] Elias N. Houstis, and John R. Rice, The engineering of modern interfaces for PDE solvers, extended abstract in *Proc. 13th IMACS World Congress*, 4, IMACS, (1991) 684–685.

[7] E.N. Houstis, J.R. Rice, N.P. Chrisochoides, H.C. Karathanasis, P.N. Papachiou, M.K. Samartzis, E.A. Vavalis, K.-Y. Wang, and S. Weerawarana, //ELLPACK: A numerical simulation programming environment for parallel MIMD machines, *Proc. 1990 Intl. Conf. Supercomputing*, ACM Press, New York, (1990), 96–107.

[8] E.N. Houstis, J.R. Rice, P. Vavodoglou, and C.E. Houstis, The PYTHIA expert system for parallel ELLPACK, Computer Science Dept., Purdue University, (1992).

[9] John R. Rice and Ronald F. Boisvert, *Solving Elliptic Problems Using ELLPACK*, Springer-Verlag, New York, 1985.